

# UTILITY PATENT APPLICATION TRANSMITTAL (Large Entity)

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No.  
AM9-99-0157

Total Pages in this Submission

## TO THE ASSISTANT COMMISSIONER FOR PATENTS

Box Patent Application  
Washington, D.C. 20231

Transmitted herewith for filing under 35 U.S.C. 111(a) and 37 C.F.R. 1.53(b) is a new utility patent application for an invention entitled:

**METHOD FOR GENERATION OF AN N-WORD PHRASE DICTIONARY FROM A TEXT CORPUS**

and invented by:

- 1) Jeffrey Thomas Kreulen
- 2) William Scott Spangler

If a **CONTINUATION APPLICATION**, check appropriate box and supply the requisite information:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: \_\_\_\_\_

Which is a:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: \_\_\_\_\_

Which is a:

☐ Continuation ☐ Divisional ☐ Continuation-in-part (CIP) of prior application No.: \_\_\_\_\_

Enclosed are:

### Application Elements

1. ☒ Filing fee as calculated and transmitted as described below
2. ☒ Specification having 24 pages and including the following:
  - a. ☒ Descriptive Title of the Invention
  - b. ☐ Cross References to Related Applications (if applicable)
  - c. ☐ Statement Regarding Federally-sponsored Research/Development (if applicable)
  - d. ☐ Reference to Microfiche Appendix (if applicable)
  - e. ☒ Background of the Invention
  - f. ☒ Brief Summary of the Invention
  - g. ☒ Brief Description of the Drawings (if drawings filed)
  - h. ☒ Detailed Description
  - i. ☒ Claim(s) as Classified Below
  - j. ☒ Abstract of the Disclosure

# UTILITY PATENT APPLICATION TRANSMITTAL (Large Entity)

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No.  
AM9-99-0157

Total Pages in this Submission

## Application Elements (Continued)

3. ☒ Drawing(s) (when necessary as prescribed by 35 USC 113)
- a. ☒ Formal Number of Sheets 4 (Figs. 1-4)
- b. ☐ Informal Number of Sheets \_\_\_\_\_
4. ☒ Oath or Declaration
- a. ☒ Newly executed (original or copy) ☐ Unexecuted
- b. ☐ Copy from a prior application (37 CFR 1.63(d)) (for continuation/divisional application only)
- c. ☒ With Power of Attorney ☐ Without Power of Attorney
- d. ☐ DELETION OF INVENTOR(S)  
Signed statement attached deleting inventor(s) named in the prior application,  
see 37 C.F.R. 1.63(d)(2) and 1.33(b).
5. ☐ Incorporation By Reference (usable if Box 4b is checked)  
The entire disclosure of the prior application, from which a copy of the oath or declaration is supplied under Box 4b, is considered as being part of the disclosure of the accompanying application and is hereby incorporated by reference therein.
6. ☐ Computer Program in Microfiche (Appendix)
7. ☐ Nucleotide and/or Amino Acid Sequence Submission (if applicable, all must be included)
- a. ☐ Paper Copy
- b. ☐ Computer Readable Copy (identical to computer copy)
- c. ☐ Statement Verifying Identical Paper and Computer Readable Copy

## Accompanying Application Parts

8. ☒ Assignment Papers (cover sheet & document(s))
9. ☐ 37 CFR 3.73(B) Statement (when there is an assignee)
10. ☐ English Translation Document (if applicable)
11. ☐ Information Disclosure Statement/PTO-1449 ☐ Copies of IDS Citations
12. ☐ Preliminary Amendment
13. ☒ Acknowledgment postcard
14. ☐ Certificate of Mailing
- ☐ First Class ☐ Express Mail (Specify Label No.): \_\_\_\_\_

**UTILITY PATENT APPLICATION TRANSMITTAL  
(Large Entity)**

(Only for new nonprovisional applications under 37 CFR 1.53(b))

Docket No.  
AM9-99-0157

Total Pages in this Submission

**Accompanying Application Parts (Continued)**

15. ☐ Certified Copy of Priority Document(s) (if foreign priority is claimed)
16. ☐ Additional Enclosures (please identify below):

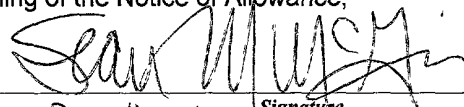
**Fee Calculation and Transmittal**

**CLAIMS AS FILED**

For	#Filed	#Allowed	#Extra	Rate	Fee
Total Claims	17	- 20 =	0	x \$18.00	\$0.00
Indep. Claims	3	- 3 =	0	x \$78.00	\$0.00
Multiple Dependent Claims (check if applicable) <input type="checkbox"/>					\$0.00
BASIC FEE					\$690.00
OTHER FEE (specify purpose) Assignment Recordation					\$40.00
TOTAL FILING FEE					\$730.00

- ☒ A check in the amount of \$730.00 to cover the filing fee is enclosed.
- ☒ The Commissioner is hereby authorized to charge and credit Deposit Account No. 50-0481 as described below. A duplicate copy of this sheet is enclosed.
- ☐ Charge the amount of as filing fee.
- ☒ Credit any overpayment.
- ☒ Charge any additional filing fees required under 37 C.F.R. 1.16 and 1.17.
- ☐ Charge the issue fee set in 37 C.F.R. 1.18 at the mailing of the Notice of Allowance, pursuant to 37 C.F.R. 1.311(b).

Dated: July 31, 2000

  
Reg. # 34386  
Frederick W. Gibbs, III  
Reg. No. 37,629

CC:

**McGINN & GIBB, P.C.**  
**A PROFESSIONAL LIMITED LIABILITY COMPANY**  
**PATENTS, TRADEMARKS, COPYRIGHTS, AND INTELLECTUAL PROPERTY LAW**  
**1701 CLARENDON BOULEVARD, SUITE 100**  
**ARLINGTON, VIRGINIA 22209**  
**TELEPHONE (703) 294-6699**  
**FACSIMILE (703) 294-6696**

**APPLICATION  
FOR  
UNITED STATES  
LETTERS PATENT**

**APPLICANT:**       **Jeffrey Thomas Kreulen**  
                              **William Scott Spangler**

**FOR:**               **"METHOD FOR GENERATION OF AN**  
                              **N-WORD PHRASE DICTIONARY**  
                              **FROM A TEXT CORPUS"**

**DOCKET NO.:**       **AM9-99-0157**

# **METHOD FOR GENERATION OF AN N-WORD PHRASE DICTIONARY FROM A TEXT CORPUS**

## **BACKGROUND OF THE INVENTION**

### *Field of the Invention*

5           The present invention generally relates to automated document clustering, and more particularly to a system and method for creating word and phrase dictionaries that are based upon the word frequency of text documents.

### *Description of the Related Art*

10           Automated document clustering is a key technology for grouping on-line text documents, such as those found on the Internet. Document clustering algorithms typically represent each document as an attribute vector, where each position of the vector represents the word frequency of a dictionary term.

15           Conventional systems for generating a dictionary from a text corpus have focused on individual words or have generated phrases based on a linguistic analysis. This conventional process is substantially more complex than the invention, as discussed below. Conventional methodologies do not describe a space and time efficient implementation for discovering phrases. As discussed in greater detail below, the invention is designed to quickly create a dictionary of

maximal frequency terms (and/or phrases) using the smallest possible amount of memory.

## SUMMARY OF THE INVENTION

5           It is, therefore, an object of the present invention to provide a structure and method for automatically creating a dictionary for clustering text documents, including performing a first pass for each of the documents to determine a frequency of each word in each of the documents, creating a Hashtable of most frequently occurring words in the documents, performing a second pass for each of  
10       the documents to determine a frequency of phrases in each of the documents that contain only words in the Hashtable and adding the most frequently occurring phrases to the Hashtable, and outputting the most frequently occurring words and the most frequently occurring phrases as the dictionary. The determination of the frequency of each word can include removing punctuation and case from the  
15       documents, removing stop words from the document, replacing words in the documents with synonyms, removing duplicate words from the documents, adding remaining words to the Hashtable, determining the frequency of each word remaining in the Hashtable, and removing words below a frequency level from the Hashtable.

20           Determining a frequency of phrases can also include removing punctuation and case from the documents, removing stop words from the documents, replacing words in the documents with synonyms, adding the phrases in each of

the documents that contain only words in the Hashtable to the Hashtable, determining the frequency of the phrases remaining in the Hashtable, and removing phrases below a frequency level from the Hashtable.

## **BRIEF DESCRIPTION OF THE DRAWINGS**

5           The foregoing and other objects, aspects and advantages will be better understood from the following detailed description of a preferred embodiment of the invention with reference to the drawings, in which:

Figure 1 is a flow diagram illustrating a first embodiment of the invention;

10           Figure 2 is a flow diagram illustrating, in greater detail, an item shown in Figure 1;

Figure 3 is a flow diagram illustrating, in greater detail, an item shown in Figure 1; and

Figure 4 it is a schematic diagram of a hardware embodiment of the invention.

## **15           DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS OF THE INVENTION**

The invention comprises a process for creating a dictionary for use in a vector representation of a document corpus. In particular, the invention uses a two pass approach to discover not only single words, but also N-word phrases,

where  $N$  is an integer greater than one. The implementation of this invention can discover all of the most commonly occurring phrases in a text corpus in a time and memory efficient manner.

The invention allows the user to specify the size of the vector space model to be used in text clustering of a document corpus, as well as the maximum number of words that can occur in a phrase. The invention will find all of the phrases, up to the user specified length, that occur with the greatest frequency. The total number of phrases returned will depend upon the user specified maximum dictionary size.

The user inputs, for a given set of text documents, a value for phrase length ( $N$ ), a vector space size ( $V$ ), an optional set of stop words, and a table of synonyms, the invention finds those phrases that occur most frequently in the set of text documents.

More specifically, referring to Figure 1, the invention performs a first pass on the set of text documents, as shown in the item 10. The detailed operations occurring in item 10 are explained below with respect to Figure 2. Next, in item 11, the invention creates a Hashtable and keeps only the most frequently occurring words in the Hashtable. More specifically, the invention finds the  $V$  most frequently occurring words in the word-count Hashtable and conserves memory by removing from the Hashtable all words that occur with less frequency than the  $V$  most frequently occurring words. Then, as shown in item 12, the invention performs a second pass on the input set of text documents. The detailed operation of item 12 is shown in Figure 3 and is discussed in greater detail below. In item



13, the invention adds phrases that are made up only of words in the word-count  
Hashtable to a phrase-count Hashtable. Finally, in item 14, the invention finds the  
most frequently occurring V words and phrases in the Hashtables and creates a  
dictionary of words and phrases from the Hashtables. While two separate  
5 Hashtables are discussed above, as would be known by one ordinarily skilled in  
the art given this disclosure, a single combined Hashtable could be utilized by the  
invention.

As shown in Figure 2, for all documents in the input set (pass 1, item 10),  
punctuation is removed and all letters are converted to lower case (item 20).

10 Next, as shown in item 21, stop words are removed and any words occurring in  
the synonym table are replaced with the designated synonym (item 22). Also,  
duplicated words are removed in item 23. The count of each of the remaining  
words are incremented in the word-count Hashtable (item 24).

Referring now to Figure 3, for all documents in the input set (pass 2, item  
15 12), the invention again removes punctuation and converts all characters to lower  
case (item 30). In item 31, stop words are removed and in item 32, words  
occurring in the synonym table are replaced with the designated synonym.

As shown in item 33, the invention determines which phrases (of two or  
more consecutive words) are made up only of words that are in the word-count  
20 Hashtable. More specifically, for x words ( $x = 2; x \leq N; x++$ ), the invention  
adds phrases to the phrase-count Hashtable and increments the count (in the  
phrase-count Hashtable) of each unique phrase of x words occurring in sequences  
in the word list, if and only if all x words are contained in the word-count

Hashtable. In order to conserve memory, the objects contained in the phrase-count Hashtable are preferably pointers to existing objects in the word-count Hashtable. Duplications of the same phrase are not counted.

The inventive process for finding phrases in a text document can be illustrated with the following example. Assume the text corpus consists of only two documents: 1. "The quick, brown fox jumped over the lazy dog." 2. "There is nothing worse than a lazy dog, except a speedy, brown fox." Further, in this example,  $N = 3$  (max number of words per phrase) and that  $V = 5$ , (desired dictionary size). Further assume that the stop word list contains the words (the, and, a, there, is, than), while the synonym table contains the entries (quick & speedy = fast, and jumped & jumping = jump).

The invention performs the first pass on the first document, as shown in item 10 in Figure 1 and shown in items 20-24 of Figure 2. The punctuation is removed (item 20 Figure 2) and the first document produces the list (the quick brown fox jumped over the lazy dog). Stop words are removed (item 21) to produce the list (quick brown fox jumped over lazy dog). Next, synonyms are replaced (item 22) to produces the list (fast brown fox jump over lazy dog). Duplicate words are removed (item 23) to produce the list (fast brown fox jump over lazy dog). The word-count Hashtable is incremented (item 24) to produce the word-count Hashtable (fast 1), (brown 1), (fox 1), (jump 1), (over 1), (lazy 1), (dog 1).

The same process is repeated on the second document "There is nothing worse than a lazy dog, except a speedy, brown fox." As discussed above, items

20-23 are applied to the second document such that the punctuation is removed, stop words are removed, synonyms are replaced, and duplicate words are removed to produce the list (nothing worse lazy dog except fast brown fox). The word-count Hashtable is then modified (item 24) to produce the following word counts (fast 2), (brown 2), (fox 2), (jump 1), (over 1), (lazy 2), (dog 2), (nothing 1), (worse 1), (except 1). The processing of the first and second documents can be done sequentially or in parallel.

As shown in item 11 in Figure 1, only the five most frequent words are allowed to remain in the Hashtable (as limited by the desired dictionary size of 5 words). In this example, only the five most frequently occurring words are fast, brown, fox, lazy, and dog.

Then, the invention performs a second pass on the first document, as shown in item 12 in Figure 1 and, as shown in greater detail in Figure 3. Once again, the punctuation and stop words are removed (items 30, 31) and the synonyms are replaced (item 32) to produce the list (fast brown fox jump over lazy dog). Then, as shown in item 33, the invention determines which phrases are made up only of words that are in the word-count Hashtable, which produces the list (fast brown), (brown fox), (lazy dog) for  $x=2$  and (fast brown fox) for  $x=3$ . Phrases such as "fox jump" are not counted because jump is not contained in the word-count Hashtable.

The same process is repeated on the second document which produces the list (lazy dog), (fast brown), (brown fox) for  $x=2$  and (fast brown fox) for  $x=3$ . After these additions, the phrase-count Hashtable stands as follows (fast-brown 2),

(brown-fox 2), (lazy-dog 2), (fast-brown-fox 2). Then, in item 14, the invention produces the resulting list containing the most frequently occurring words and phrases: (fast, brown, fox, lazy, dog, fast-brown, brown-fox, lazy-dog, fast-brown-fox)

5           This invention can be implemented utilizing any conventional programming language and math. For example as a computer program, written in the Java programming language and executed with the Java virtual machine could be used, as shown below:

```

10           Hashtable allWords = new Hashtable();
          Hashtable allPhrases = new Hashtable();
          Hashtable stopWords = new Hashtable();
          Hashtable synonyms = new Hashtable();

          String words[] = null; // this is the resulting dictionary

15   Public PhraseDictionary(String textfile,
         String stopWordsFile, String synonymsFile, nt N, int V) {

          // read in stop words and put them in a hashtable.
          try {
              if (stopWordsFile!=null) {
                  BufferedReader br = Util.openReadFile(stopWordsFile);
20               while (true) {
                      String word = br.readLine();
                      if (word==null // word.equals("")) break;
                      StopWords.put(word.notNull); }

              }

25            }

          // read in synonyms and put them in a hashtable.

          if (synonymsFile!=null) {
              BufferedReader br = Util.openReadFile(synonymsFile);
              while (true) {
30               String line = br.readLine();
              if (line==null // line.equals("")) break;

```

```

5      // a StringVector is a Vector of Strings. This creation
        // method will tokenize the input String by the 'space'
        // character, creating a list of words.
        // Each line of the input file represents a list of
        // synonyms. The synonyms hashtable provides a means of
        // replacing each of these words with the first word of the //
        line.

        StringVector sv=new StringVector(line." ");

        for (int i:1: i<sv.size(): i++)
10            Synonyms.put(sv.myElementAt(i).sv.
                myElementAt(0));

        }
    }

15    -// ndata represents the number of lines in the textfile. The
        // textfile represents all examples in the data set, one example
        // per line.

        int ndata = Util.getTextLength(textfile);

        BufferedReader br=Util.openReadFile(textfile);
        StringVector sv = null;

20    // This for loop represents the first pass of the algorithm.

        for (int i=0; i<ndata; i++) {

            // Create a Vector of words from a text example.
            sv=stringToStringVector(br.readLine());

            // Remove stop Words
25            sv = removeStopWords(sv);

            // Replace synonyms
            sv = replaceSynonyms(sv);

            // Count all words.
            parseStringForWords(sv):

30        };

```

```

// calculate how frequently a word must occur to be
// maintained

int threshold = findThreshold(V);

cleanUp(threshold);

5      br = Util.openReadFile(textile);

// Begin second pass through text data set.

for (int i:0; i<ndata; i++) {

    sv = toStringVector(br.readLine());
    sv = replaceSynonyms(sv);
10    sv = removeStopWords(sv);

    for (int x=2; x<=N; x++) {

        // Create x-word phrases from the list of
        // words
        // A Phrase object is simply an ordered list
        // of words.
15    Phrase p[] = createPhrases(sv,x);
        // Remember only phrases that use words in
        // the // allWords hashtable
        ParseStringForPhrases(p);

20    Threshold = findThreshold(V);

    StringVector savedWords = new StringVector();

    // save terms in the allWords hashtable that are greater
    // than the threshold

25    Enumeration e = allWords.keys();
    while (e.hasMoreElements()) {
        String s = (String)e.nextElement();
        WordCounter val = (WordCounter)allWords.get(s);
        if (val.wordcount>threshold) {
            savedWords.addElement(s);
30    }
    }
}

```

```

// save terms in the allPhrases hashtable that are greater
// than the threshold

        E = allPhrases.keys();
        while (e.hasMoreElements()) {
5             Phrase s = (Phrase)e.nextElement();
              WordCounter val =
                  (WordCounter)allPhrases.get(s);
              if (val.wordcount>threshold) {
                  savedWords.addElement(""+s);
10             }
        }

        // convert the vector to an array

        Words = savedWords.getStringArray();

    } catch (Exception e) {e.printStackTrace();};
15 }

public static StringVector stringToStringVector(String s) {

    s = s.toLowerCase();
    StringBuffer sb = new StringBuffer(2000);
    -int begin2 = (int)'a';
20    int end2 = (int)'z';
    int begin3 =(int)'0';
    int end3 = (int)'9';

    for (int i:0; i<s.length(); i++) {
        int c = s.charAt(i);
25        if (c>=begin2 && c<=end2) {
            sb.append((char)c);

        }

        else if (c>=begin3 && c<=end3)
            sb.append((char)c);
30        else sb.append(' ');
    }
    String stuff = new String(sb);
    StringVector result = new StringVector(stuff," ");
    return(result);
35 }

```

```

Public StringVector replaceSynonyms(StringVector sv) {
    StringVector result = new StringVector();
    for (int i=0; i<sv.size(); i++) {
        String s = sv.myElementAt(i);
        String syn = (String)synonyms.get(s);
        if (syn==null) result.addElement(s);
        else result.addElement(syn);
    }
    return(result);
}

public StringVector removeStopWords(StringVector sv) {
    StringVector result = new StringVector();
    for (int i=0; i<sv.size(); i++) {
        String s = sv.myElementAt(i);
        if (stopWords.get(s)==null) result.addElement(s);
    }
    return(result);
}

public void parseStringForWords(StringVector sv) {
    sv = removeDuplicates(sv);
    for (int i=0; i<sv.size(); i++) {
        String s = sv.myElementAt(i);
        Object temp = stopWords.get(s);
        if (temp!=null) continue;
        WordCounter val = (WordCounter)allWords.get(s);
        if (val==null) {
            val = new WordCounter(s);
            allWords.put(s,val);
        }
        else val.inc();
    }
}

public int findThreshold(int numWords) {
    int n = allWords.size();
    int m = allPhrases.size();
    int wcounts[] = new int[n];
    int pcounts[] = new int[m];
    Enumeration e = allWords.elements();
    for (int i=0; i<n; i++) {

```



```

        wordCounter wc = (WordCounter)e.nextElement();
        wcounts[i] = wc.wordcount;

    }

    e = allPhrases.elements();
5   for (int i=0; i<m; i++) {
        wordCounter wc = (WordCounter)e.nextElement();
        pcounts[i] = wc.wordcount;

    }

    int worder[] = Index.run(wcounts);
10   int porder[] = Index.run(pcounts);
    int wpos = wcounts.length-1;
    int ppos = pcounts.length-1;
    int total = 0;
    while (total<numWords) {
15         if (wcounts[worder[wpos]]<pcounts[porder[ppos]])
            Ppos--;
        else wpos--;
        Total++;
        if (ppos==0 // wpos == 0) break;
20     }
    return(wcounts[worder[wpos]]);
}

public void cleanUp(int i) {
    Enumeration e = allWords.keys();
25   while (e.hasMoreElements()) {
        String s = (String)e.nextElement();
        WordCounter val = (WordCounter)allWords.get(s);
        if (val.wordcount<=i){
            allWords.remove(s);
30         }
    }
    e = allPhrases.keys();
    while (e.hasMoreElements()) {
        Phrase s = (Phrase)e.nextElement();
35   WordCounter val = (WordCounter)allPhrases.get(s);
        if (val.wordcount<=i) {
            allPhrases.remove(s);
        }
    }
40 }

```

```

public Phrase[] createPhrases(StringVector sv, int size) {
    if (sv.size() < size) return(new Phrase[0]);
    Phrase[] result = new Phrase[sv.size() - (size - 1)];
    for (int i = 0; i < result.length; i++) {
5        String s[] = new String[size];
            for (int j = 0; j < s.length; j++) {
                s[j] = sv.elementAt(i + j);
            }
            result[i] = new Phrase(s);
10        }
        Return(result);
    }

    public void parseStringForPhrases(Phrase p[]) {
        MyIntVector duplicates = new MyIntVector();
15        for (int i = 1; i < p.length; i++) {
            for (int j = 0; j < i; j++) {
                if (p[j].equals(p[i])) duplicates.addElement(i);
            }
        }
20        for (int i = 0; i < p.length; i++) {

            // ignore duplicate phrases
            if (duplicates.myContains(i)) continue

            // ignore phrases having duplicate words
            if (p[i].containsDuplicates()) continue;

25            // ignore phrases which have a word not in dictionary
            for (int j = 0; j < p[i].length; j++) {
                if (allWords.get(p[i].nth(j)) == null) continue;
            }
            WordCounter val = (WordCounter) allPhrases.get(p[i]);
30            if (val == null) {
                val = new WordCounter();
                allPhrases.put(p[i], val);
            }
            else val.inc();
35        }
    }
}

```

Many existing methods for generating a dictionary from a text corpus have focused on individual words only or have generated phrases based on a linguistic analysis. The invention's methodology is purely lexical in nature and thus generalizes to multiple languages and to ungrammatical text. Previous methodologies that have suggested a lexical phrase generation technique have not described the space and time efficient implementation for discovering such phrases that the invention utilizes. The invention's implementation is designed to quickly find a maximal frequency term dictionary of a given size using the smallest possible amount of memory.

While the overall methodology of the invention is described above, the invention can be embodied in any number of different types of systems and executed in any number of different ways, as would be known by one ordinarily skilled in the art. For example, as illustrated in Figure 4, a typical hardware configuration of an information handling/computer system in accordance with the invention preferably has at least one processor or central processing unit (CPU) 400. For example, the central processing unit 400 could include various image/texture processing units, mapping units, weighting units, classification units, clustering units, filters, adders, subtractors, comparators, etc. Alternatively, as would be known by one ordinarily skilled in the art given this disclosure, multiple specialized CPU's (or other similar individual functional units) could perform the same processing, mapping, weighting, classifying, clustering, filtering, adding, subtracting, comparing, etc.

The CPU 400 is interconnected via a system bus 401 to a random access memory (RAM) 402, read-only memory (ROM) 403, input/output (I/O) adapter 404 (for connecting peripheral devices such as disk units 405 and tape drives 406 to the bus 401), communication adapter 407 (for connecting an information handling system to a data processing network) user interface adapter 408 (for connecting peripherals 409-410 such as a keyboard, mouse, imager, microphone, speaker and/or other interface device to the bus 401), a printer 411, and display adapter 412 (for connecting the bus 401 to a display device 413). The invention could be implemented using the structure shown in Figure 4 by including the inventive method, described above, within a computer program stored on the storage device 405. Such a computer program would act on an image supplied through the interface units 409-410 or through the network connection 407. The system would then automatically segment the textures and output the same on the display 413, through the printer 411 or back to the network 407.

The benefits which flow from this invention are derived from the ability to readily adapt the creation of text dictionaries containing both words and phrases to the capabilities of the computer hardware available. The invention allows the user to specify the dictionary size up front, without reference to the size or complexity of the data set to be analyzed, and the invention returns all of the most frequent terms which can fit within this memory constraint. This allows the user to analyze text data sets of arbitrary size and complexity on computer hardware of fixed memory and computational speed. Creation of word/phrase dictionaries on text data sets further allows for the analysis of unstructured text information in a

semi-structured manner. Data mining algorithms and statistical measure can now be applied to the data to discover interesting relationships and trends. Dictionary creation is thus the first critical step in data mining and analysis of text data sets. Being able to generate such dictionaries quickly and efficiently and with high quality is therefore of key importance to successful text mining.

While the invention has been described in terms of preferred embodiments, those skilled in the art will recognize that the invention can be practiced with modification within the spirit and scope of the appended claims.

## CLAIMS

What is claimed is:

1. A method of automatically creating a dictionary for clustering text documents comprising:

5 determining a frequency of each word in each of said documents;  
creating a Hashtable of most frequently occurring words in said documents;  
determining a frequency of phrases in each of said documents that contain only words in said Hashtable;  
adding most frequently occurring phrases to said Hashtable; and  
10 outputting said most frequently occurring words and said most frequently occurring phrases as said dictionary.

2. The method in claim 1, wherein said determining a frequency of each word comprises:

removing punctuation and case from said documents;  
15 removing stop words from said document;  
replacing words in said documents with synonyms;  
removing duplicate words from said documents;  
adding remaining words to said Hashtable;  
determining said frequency of each word remaining in said Hashtable; and  
20 removing words below a frequency level from said Hashtable.

3. The method in claim 2, further comprising inputting one or more of said stop words, said synonyms, and said frequency level.

4. The method in claim 1, wherein said determining a frequency of phrases comprises:

- 5 removing punctuation and case from said documents;
- removing stop words from said document;
- replacing words in said documents with synonyms;
- adding said phrases in each of said documents that contain only words in said Hashtable to said Hashtable;
- 10 determining said frequency of said phrases remaining in said Hashtable;
- and
- removing phrases below a frequency level from said Hashtable.

5. The method in claim 4, further comprising inputting one or more of said stop words, said synonyms, and said frequency level.

15 6. A method of automatically creating a dictionary for clustering text documents comprising:

- performing a first pass for each of said documents comprising:
  - determining a frequency of each word in each of said documents;
- and

creating a Hashtable of most frequently occurring words in said documents;

performing a second pass for each of said documents comprising:

5 determining a frequency of phrases in each of said documents that contain only words in said Hashtable; and

adding most frequently occurring phrases to said Hashtable; and  
outputting said most frequently occurring words and said most frequently occurring phrases as said dictionary.

10 7. The method in claim 6, wherein said determining a frequency of each word comprises:

removing punctuation and case from said documents;

removing stop words from said document;

replacing words in said documents with synonyms;

removing duplicate words from said documents;

15 adding remaining words to said Hashtable;

determining said frequency of each word remaining in said Hashtable; and

removing words below a frequency level from said Hashtable.

8. The method in claim 7, further comprising inputting one or more of said stop words, said synonyms, and said frequency level.



9. The method in claim 6, wherein said determining a frequency of phrases comprises:

removing punctuation and case from said documents;

removing stop words from said document;

5 replacing words in said documents with synonyms;

adding said phrases in each of said documents that contain only words in said Hashtable to said Hashtable;

determining said frequency of said phrases remaining in said Hashtable;

and

10 removing phrases below a frequency level from said Hashtable.

10. The method in claim 9, further comprising inputting one or more of said stop words, said synonyms, and said frequency level.

11. A program storage device readable by machine, tangibly embodying a program of instructions executable by the machine to perform a method of

15 automatically creating a dictionary for clustering text documents, said method comprising:

determining a frequency of each word in each of said documents;

creating a Hashtable of most frequently occurring words in said documents;

determining a frequency of phrases in each of said documents that contain

20 only words in said Hashtable;

adding most frequently occurring phrases to said Hashtable; and

outputting said most frequently occurring words and said most frequently occurring phrases as said dictionary.

12. A program storage device as in claim 11, wherein said determining a frequency of each word comprises:

- 5 removing punctuation and case from said documents;
- removing stop words from said document;
- replacing words in said documents with synonyms;
- removing duplicate words from said documents;
- adding remaining words to said Hashtable;
- 10 determining said frequency of each word remaining in said Hashtable; and
- removing words below a frequency level from said Hashtable.

13. A program storage device as in claim 12, further comprising inputting one or more of said stop words, said synonyms, and said frequency level.

14. A program storage device as in claim 11, wherein said determining a frequency of phrases comprises:

- removing punctuation and case from said documents;
- removing stop words from said document;
- replacing words in said documents with synonyms;
- adding said phrases in each of said documents that contain only words in
- 20 said Hashtable to said Hashtable;

determining said frequency of said phrases remaining in said Hashtable;

and

removing phrases below a frequency level from said Hashtable.

15. A program storage device as in claim 14, further comprising inputting said

5 stop words.

16. A program storage device as in claim 14, further comprising inputting said

synonyms.

17. A program storage device as in claim 14, further comprising inputting said

frequency level.

# **METHOD FOR GENERATION OF AN N-WORD PHRASE DICTIONARY FROM A TEXT CORPUS**

## **ABSTRACT**

It is, therefore, an object of the present invention to provide a structure and  
5 method for automatically creating a dictionary for clustering text documents,  
including performing a first pass for each of the documents to determine a  
frequency of each word in each of the documents, creating a Hashtable of most  
frequently occurring words in the documents, performing a second pass for each of  
the documents to determine a frequency of phrases in each of the documents that  
10 contain only words in the Hashtable and adding the most frequently occurring  
phrases to the Hashtable, and outputting the most frequently occurring words and  
the most frequently occurring phrases as the dictionary. The determination of the  
frequency of each word can include removing punctuation and case from the  
documents, removing stop words from the document, replacing words in the  
15 documents with synonyms, removing duplicate words from the documents, adding  
remaining words to the Hashtable, determining the frequency of each word  
remaining in the Hashtable, and removing words below a frequency level from the  
Hashtable.

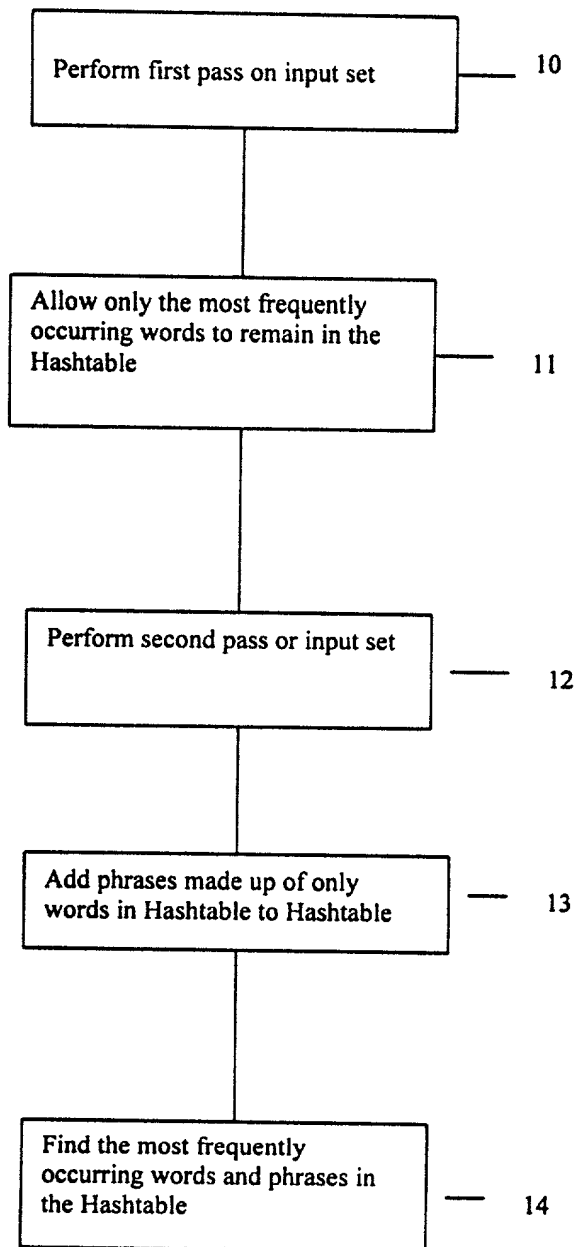


FIGURE 1

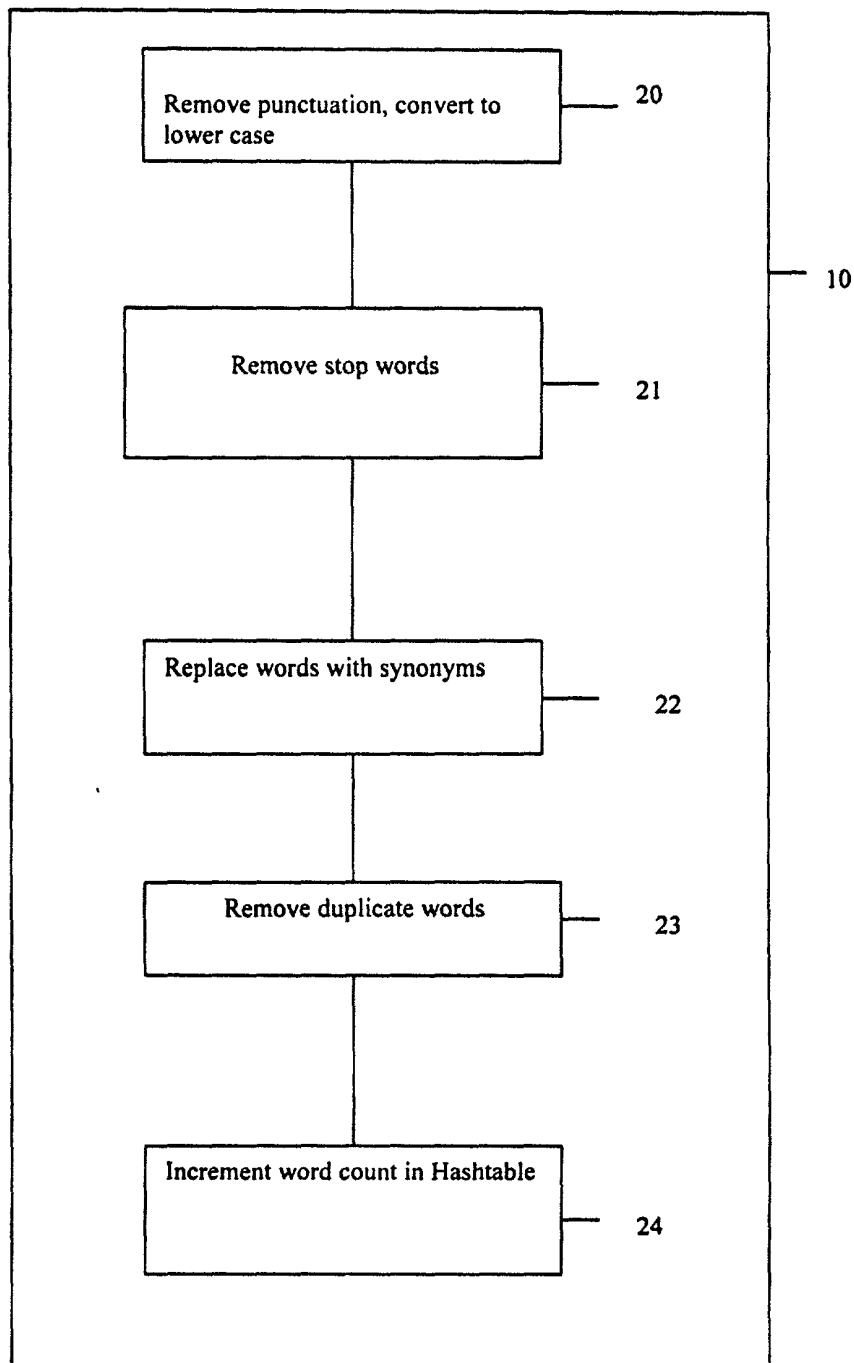


FIGURE 2

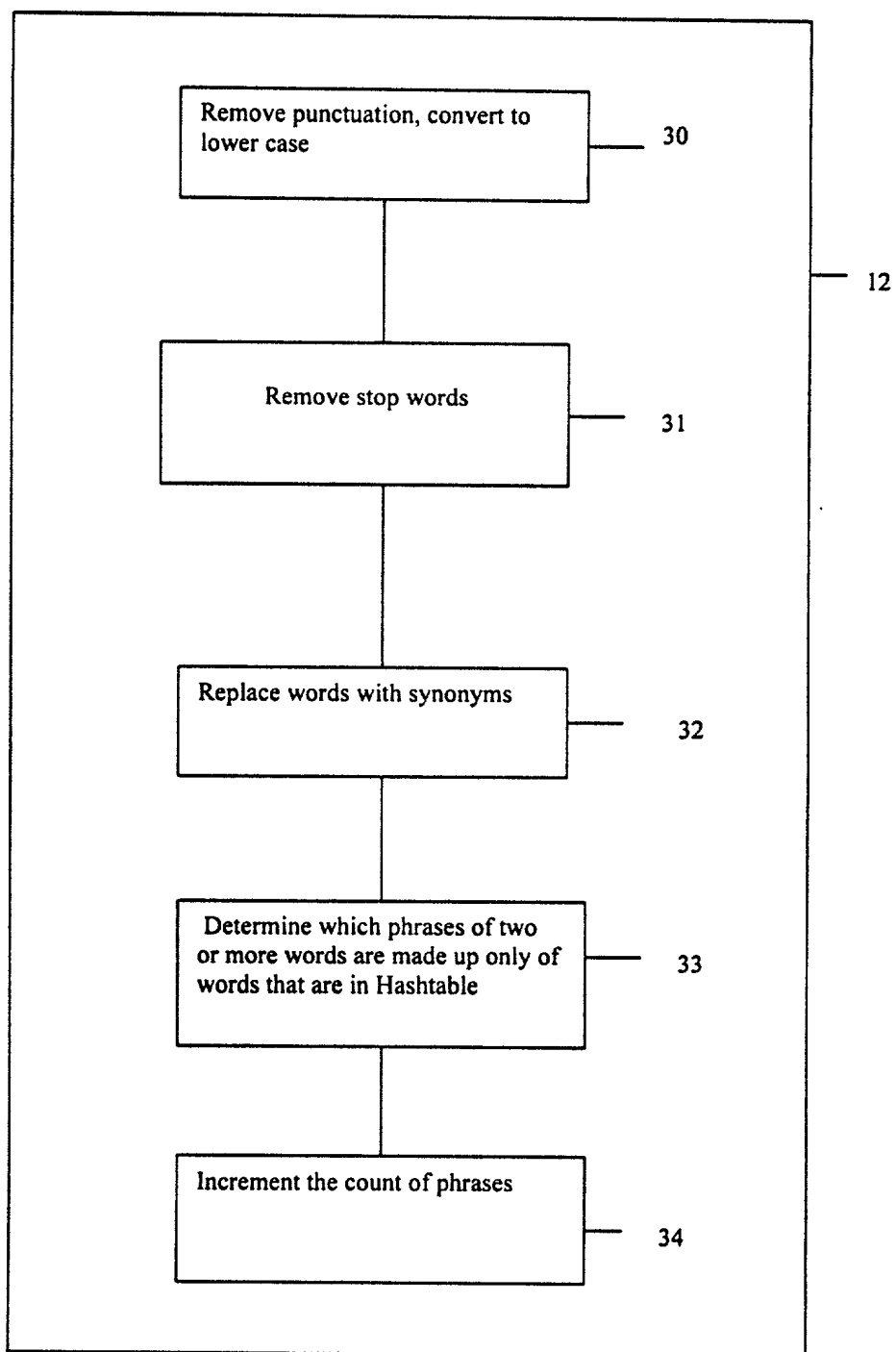


FIGURE 3

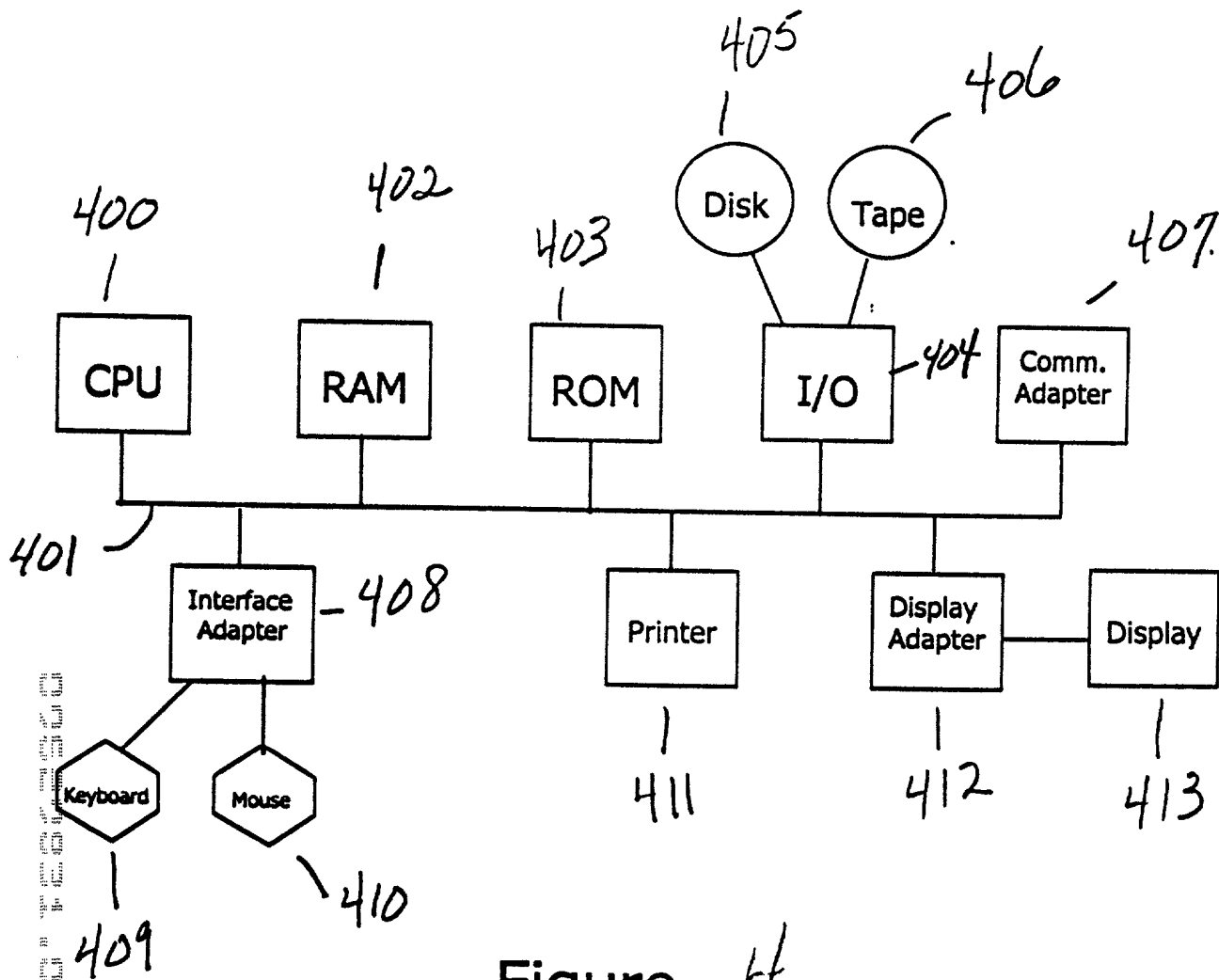


Figure 4



## DECLARATION AND POWER OF ATTORNEY

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name; I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled: METHOD FOR GENERATION OF AN N-WORD PHRASE DICTIONARY FROM A TEXT CORPUS

the specification of which:  
(check one)

- ☒ is attached hereto.  
☐ was filed on \_\_\_\_\_, as Application Serial No. \_\_\_\_\_ and was amended on \_\_\_\_\_.

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the patentability of this application in accordance with Title 37, Code of Federal Regulations, § 1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, § 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

### Prior Foreign Application(s):

Number	Country	Day/Month/Year	Priority Claimed
--------	---------	----------------	------------------

I hereby claim the benefit under Title 35, United States Code, § 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, § 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, § 1.56 which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

### Prior U.S. Applications:

Serial No.	Filing Date	Status
------------	-------------	--------

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

As a named inventor, I hereby appoint the following attorneys and/or agents to prosecute this application and transact all business in the Patent and Trademark Office connected therewith: We hereby appoint Khanh Q. Tran, Registration No. 41,352, Thomas R. Berthold, Registration No. 28,689, Marc McSwain, Registration No. 44,929, Alison D. Mortinger, Registration No. 39,306, Richard Ludwin, Registration No. 33,010, Frederick W. Gibb, III, Registration No. 37,629, and Sean M. McGinn, Registration No. 34,386 to prosecute this application and transact all business in the United States Patent and Trademark Office connected therewith.

Send all correspondence to: McGinn & Gibb, P.C., 1701 Clarendon Boulevard, Suite 100, Arlington, Virginia 22209. Customer No. 21254

Telephone calls should be directed to Sean M. McGinn, McGinn & Gibb, P.C. at (703) 294-6699.

(1) Inventor: Jeffrey Thomas Kreulen

Signature: Jeffrey J Kreulen

Date: 7/25/2000

Residence: 7220 Golf Course Lane, San Jose, CA 95139

Citizenship: United States of America

Post Office Address: Same as Residence

(2) Inventor: William Scott Spangler

Signature: William Scott Spangler

Date: 7/21/00

Residence: 12840 Stevens Court, San Martin, CA 95046

Citizenship: United States of America

Post Office Address: Same as Residence